The new WinDBG
The tool for hardcore debugging

André Vachon
Development Lead
Windows Debugger Team

Contact Info

- **Web site:**
  - http://www.microsoft.com/ddk/debugging
- **Problems, Questions, Feature Requests**
  - **Help file (Debugger.chm) is in the debugger package**
  - mailto:windbgfb@microsoft.com

Goals

- What is WinDBG\cdb ?
  - Why do they exist
  - What are the features and limitations of the debugger
  - How can it help solve problems
- Go over key OS features that help with debugging

Non-Goals

- Teach you how to debug your app

Agenda

- Symbols
  - Symbol server
- Dump files
- Debugger Architecture
  - Debugger engine, WinDBG, KD
  - Debugger Commands
- Remote Debugging
- 64 bit debugging
- Debugger extensions

Symbols

What Are .pdb Files ?

- Generated by the compiler and linker
- Binary file with multiple “streams”
  - Each streams contains a certain type of debugging information
- Matched to the executable with unique identifiers
  - Age and signature with VC6
  - Adding GUID for VC7

Full .pdb Files

- .pdb file as generated by the linker
- Has all data streams
  - Types, locals, globals, statics, fpo, fixups, OMAP, source lines, etc
- Required to do full source level debugging
- Files are very large
  - ntoskrnl.pdb is 9 MEG
- Not shipped outside Microsoft

Public .pdb Files

- Generated by binplace from the full .pdb
- Certain data streams are removed
  - Types, locals, source lines
- All function names and data required for stack traces are included
  - Globals, fpo
- Files are much smaller
  - Ntoskrnl.pdb is 1 MEG
- Public .pdb files are shipped on

Type Information

- Type information is contained in one of the .pdb streams
  - Full encoding of every type definition used in a binary
  - Name and type of each field
- Debugger can query type definitions
  - Used for structure expansion in debugger
- Not stored in the .pdb
  - #define, Unused types

Optimizers Eat Information

- Source lines confused
  - Line does not exist, overlap:
    - Inline code is merged
    - Duplicate functions removed from the code
  - Lines execute out of order
    - Gets much worse on IA64
- Locals, stack are different
  - Stack locations \ registers are reused
    - Local variable locations are wrong
- Turn-off optimizer for source

BBT'ed symbols

- Source code is moved around
  - Change the flow of execution
- OMAP
  - Pdb table that describes code movement
- <PERF> in a symbol name warns about inaccurate \ non-existent symbol
- <Function_name+offset> != <Function_address+offset>
  - PreBBT vs PostBBT

Other Symbols

- Export Symbols
  - Debugger can use public exports as symbols
  - Used when no symbol file can be found
- Sym files
  - Only used on Win9x
  - Not supported by kd\ntsd\WinDBG at this point

Symbol Server

- Symbol files indexed on a one machine
  - Use unique identifiers from the images and symbol files to find the symbols
- Symbol handler supports new lookup
  - Special symbol path pointing to the server
- Distribute tools to build a symbol server
  - Symstore.exe

Other tools for symbols

- linker
  - link –dump –imports: dlls the image is linked against
  - link –dump –headers: detailed information about the image.
- cvdump: dump information about a .pdb file

Dump Files

Types of Dump Files

- Goal of dump files:
  - Get crash information from the field
- Three types of kernel dump files
  - Full , Summary , Triage
  - Generated by the kernel on a bluescreen
- Three types of user mode dump files
  - Full, Minidumps, Full-memory Minidumps

Full Kernel Dump File

- In NT4, Win2k, Whistler
- Stores all the physical memory in the machine
  - Dump is independent of virtual addresses or processes
  - Dump file can be *very* large
  - Dump will let you see application data
- Data saved to the page file
  - Dump fails if page file is not large enough
  - Page file must be on boot drive
  - Page file content is destroyed

Summary Kernel Dump File

- New in Windows 2000
- Saves resident kernel pages only
- Advantages
  - Quicker to generate at crash time
  - Provide enough information to analyze any kernel data structure
- Disadvantages
  - Can not look at any user mode

Triage Kernel Dumps

- New in windows 2000 – enabled by default on Workstation product
- 64K in size – very quick to generate
  - Contains minimal info to pin-point failure
  - Can not be used to do a full debug session
- Triage Minidump contain:
  - Context and stack of the faulting thread
  - Current thread and process data structures

User Dump Files

- Two types of dump files
  - Full dumps
  - Minidumps
  - Full-memory Minidumps

Full User Dump File

- Dump of the entire address space of the application
- Generated automatically by Dr.Watson
- Advantages
  - Lets you debug almost any fault off line
- Disadvantages
  - Can be very large and long to generate

Mini User Dump File

- Dump only contains basic information about the crash
  - All threads, with associated context and stack data
  - List of modules
- Advantages
  - Small and quick to create
- Disadvantages
  - Can only debug simple failures (AVs)

Dump file Enhancements

- Extend the “minidump” format to support full dump files
  - “Full-memory minidumps”
  - New memory stream type
- Added Comment stream
- Added Handle information stream
  - Can extract OS handle table and store it in the dump file so ! handle can be used
- More streams to come …

Using Dump Files

- Debugger can load ANY dump file
  - kd –z <dump_file> -y <sym_path>
- All debugger commands work identically to live debugging
  - Some limitations based on information available in the dump
- Non Full-memory Minidump files
  - Must specify –i <image_path>
  - Limited data in the dump file restricts which commands are valid

Using Dump File

- Advantages
  - Very simple to use with the debugger
  - Very useful for off-line debugging
  - Can get dump files from customers
  - Can send dump files to other developers
- Disadvantages
  - Kernel dump files have no adapter hardware state available
  - Can not execute any code

32 Bit Vs. 64 Bit Dump Files

- Different file formats for kernel dumps
  - Certain fields are extended to handle 64 bit addresses
- Minidump format identical on 32 / 64
- 64 bit dump files will get bigger, as address space of machine gets bigger
- The debugger just works with

Dump File Demo

Debugger Architecture

Debugger Architecture

The debuggers have been split into:

- Debugger engine (dbgeng.dll)
  - Handles all debugging activity
  - Exposes rich set of APIs to debugger UIs and extensions
- UI (windbg, cdb, ntsd, kd, etc)
  - Simple and replaceable front end that manages user input and debugger output
- Symbol Handler (dbghelp.dll)

Debugger engine

- Single DLL, with built-in support for:
  - NT4, Win2k, Whistler, Xbox (chk and free)
  - Dynamic 32 bit and 64 bit support
  - X86, Alpha, IA64, AMD64
  - User Mode and kernel mode
  - Live and Dump File debugging
  - Local and remote debugging
  - Full source level support
  - Very powerful extension API interface

Debugger engine

- All Debugger engine features are transparent to the UI
- APIs exposed through simple COM interfaces
  - Debugger engine API is remotable

kd.exe

- Windows NT Kernel debugger
  - No Win9x kernel debug support
- Simple command line UI
  - Takes all commands and send them to the debugger engine
  - Displays output from debugger engine
- Merged i386kd, alphakd and ia64kd into one

Cdb.exe and ntsd.exe

- Win32 user-mode debugger
- Simple command line UI
  - Takes all commands and send them to the debugger engine
  - Displays output from debugger engine
- Ntsd.exe is part of the OS
- CDB.exe is part of the debugger package
- CDB is NTSD without creating a window

WinDBG.exe

- User mode and kernel mode debugger
- GUI on top of the debugger engine
  - UI similar to old Windbg (total rewrite)
- Basic GUI debugger
  - All common window types (source, locals, disassembly, register, watch, command)
  - Common keystrokes (F5, F9, F10, F11)

WinDBG and VC

- WinDBG is a debugger only
  - Not a development environment
- Our first priority is debugging the OS
  - Kernel debugging
  - Service debugging
  - Bootstraping new OS \ architectures
  - Remotability
    - Most debugging in Windows team is done remotely

WinDBG and VC

- Limited UI in WinDBG
  - No Drag\Drop, no window docking, no tool-tips
- Debugger engine has a modified masm expression evaluator
  - No support for complex C and C++ expressions
  - Special syntax for dealing with structure and pointers

Debugger install

- MSI package
  - Windbg\cdb\kd
  - All debugger extensions
  - Docs
  - Misc. tools (tlist, breakin, Etc)
- Install takes less than 1 minute
- No registry key (except MSI install keys)
  - Can use “xcopy” to install
- Can have multiple copies installed simultaneously

Debugger Command window

Debugger Commands

- Command line is very powerful
  - More powerful than the UI
- Command line is parsed by the engine
  - Output for Windbg \ cdb \ ntsd is identical
- Three types of commands
  - Regular built-in commands (dd, k)
  - “Meta” or “dot” commands (.reload, .cxr)
    - Complex, built-in commands

Debugger Command details

Read the docs !!!

Displaying/Modifying Data

- Dump memory : db, dw, dd, dq
- Dump strings: da, dc, du, ds, dS
- Dump types: dt
- Dump local variables: dv
- eX: edit data
- rXXX: display or modify registers

Stack traces

- K*[n] – print stack trace
  - Kb – with arguments
  - Kv – verbose, shows frame information
  - Kp – print type of parameters
- .frame
  - Set frame to examine
- .cxr <Exception record address>
  - Reset registers to match context record
- .ecxr

Execution And Stepping

- u [address]: Unassemble
- p [count] : Step over
- t [count] : Trace into
  - tb : taken branch trace (IA64 only)
- g: Go
  - go until: g [address | symbol]
- wt : instruction trace tree

Breakpoints

- Software breakpoints: `bp <func>`
- Hardware breakpoints: `ba e1 <func>`
- bp with command: `bp<func> “command”`
- bp on source: `bp` `` `mysource.c:142` ``
- bc: clear breakpoint(s)
- bl: list breakpoints
- bd: disable breakpoint(s)

Exception commands

- sx* <exception #>|<event ID>
  - sxe : stop on first-chance
  - sxd : stop on second chance
  - sxn : output message only
  - sxi : ignore (don’t break at all
- Exceptions
  - Any NT \ win32 exception code
- Debugger Events
  - Module Load or Unload
  - Thread\Process create\destroy

Dump file commands

- .dump: generate dump files
  - .dump /m to generate minidumps
  - .dump /f to generate full dumps
- Can be used when doing live or dump file debugging

Source commands

- .lines: Turn on source line loading in the command line debuggers
  - Turned on by default for WinDBG
- l+: enable source options
  - l+l: show source on the command prompt
  - l+o: show source only when stepping
  - l+n: list source with “ln” command

Symbol commands

- ln: List nearest symbol
- .sympath
  - Changes symbol search path
  - .sympath path1;path2;path3
- !sym noisy
  - Turns on *very* verbose symbol load.
  - Very good to diagnose symbol loading problems

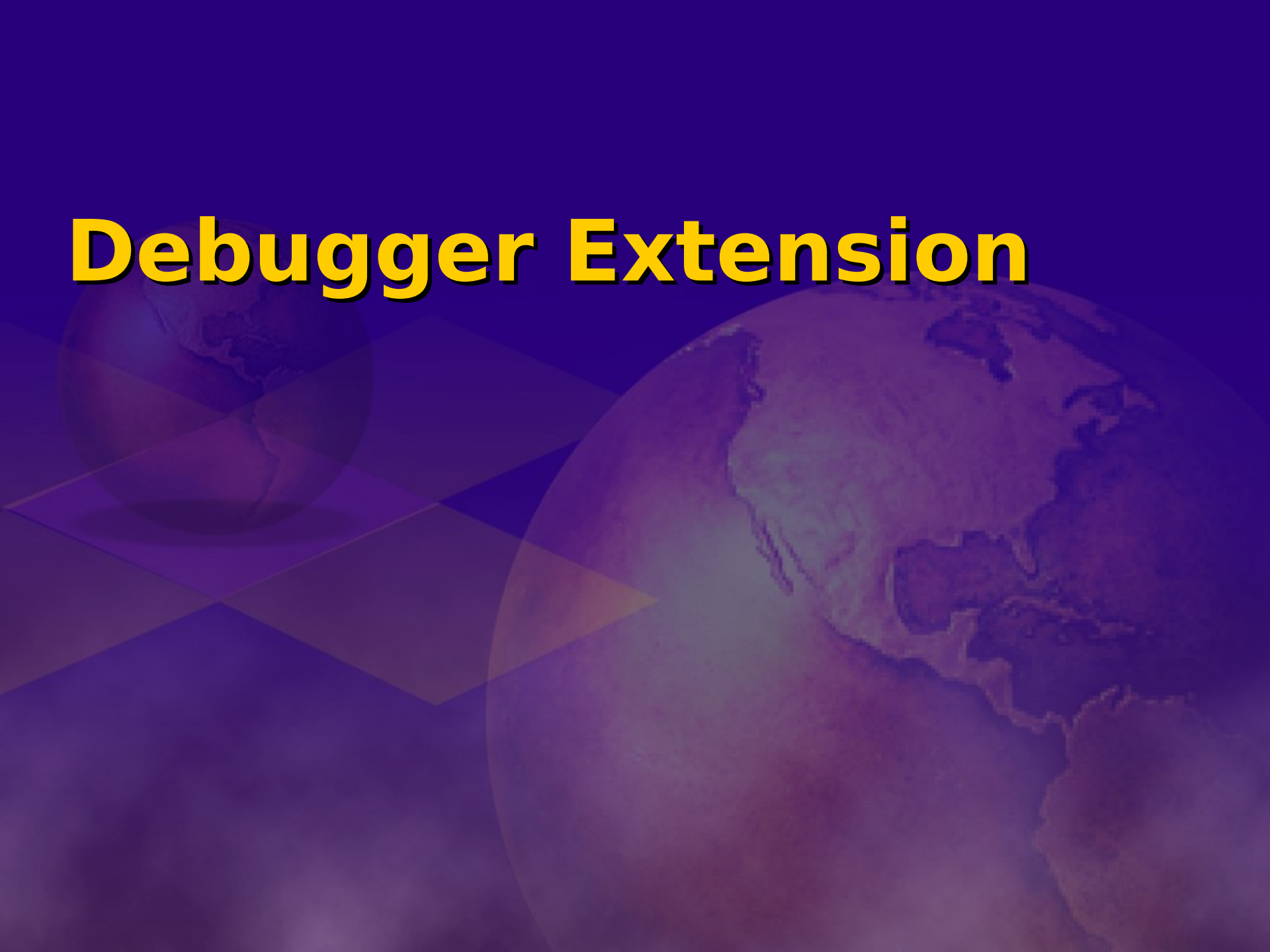
Module Commands

- .reload: Reloads all symbols
  - .reload –l: List all modules without reloading them
- lm: show module list
  - lmv: verbose module list (timestamp, etc)
  - lml: show modules with loaded symbols
  - lm mXXX*: wildcard match module names
- !dh <image_address>: dump header
  - Same output as link.dump

Diagnosing Bad Symbols

- Debugger will give you a message when symbols are not found
- !sym noisy: turn on verbose symbol load
  - “File not found” : fix with .sympath
  - “Unmatched .pdb” : verify the date on the image and .pdb file
  - Change signature of pdb file using !dh on the image address and pdbdump on the pdb.

Debugger Extension



What are debugger extensions

- Lets you write your own debugger commands
  - Code to analyze your data structures
  - Code to enumerate lists of objects
  - Code to handle specific events such as breakpoints or other events
- Windows DEV team could not debug the OS without debugger extensions

Old Debugger interfaces

- Exposed very limited functionality
  - Read and write memory
  - Thread Context
  - Expression evaluation
  - Symbol lookup

New Debugger Interfaces

- Debugger engine exposes a new, complete set of interfaces
  - Everything that can be performed by a debugger is exposed by the interface
  - WinDBG only uses debug engine interfaces for debugging operations
- Can write new standalone tools that call the interface
  - Debugger extension DLLs are

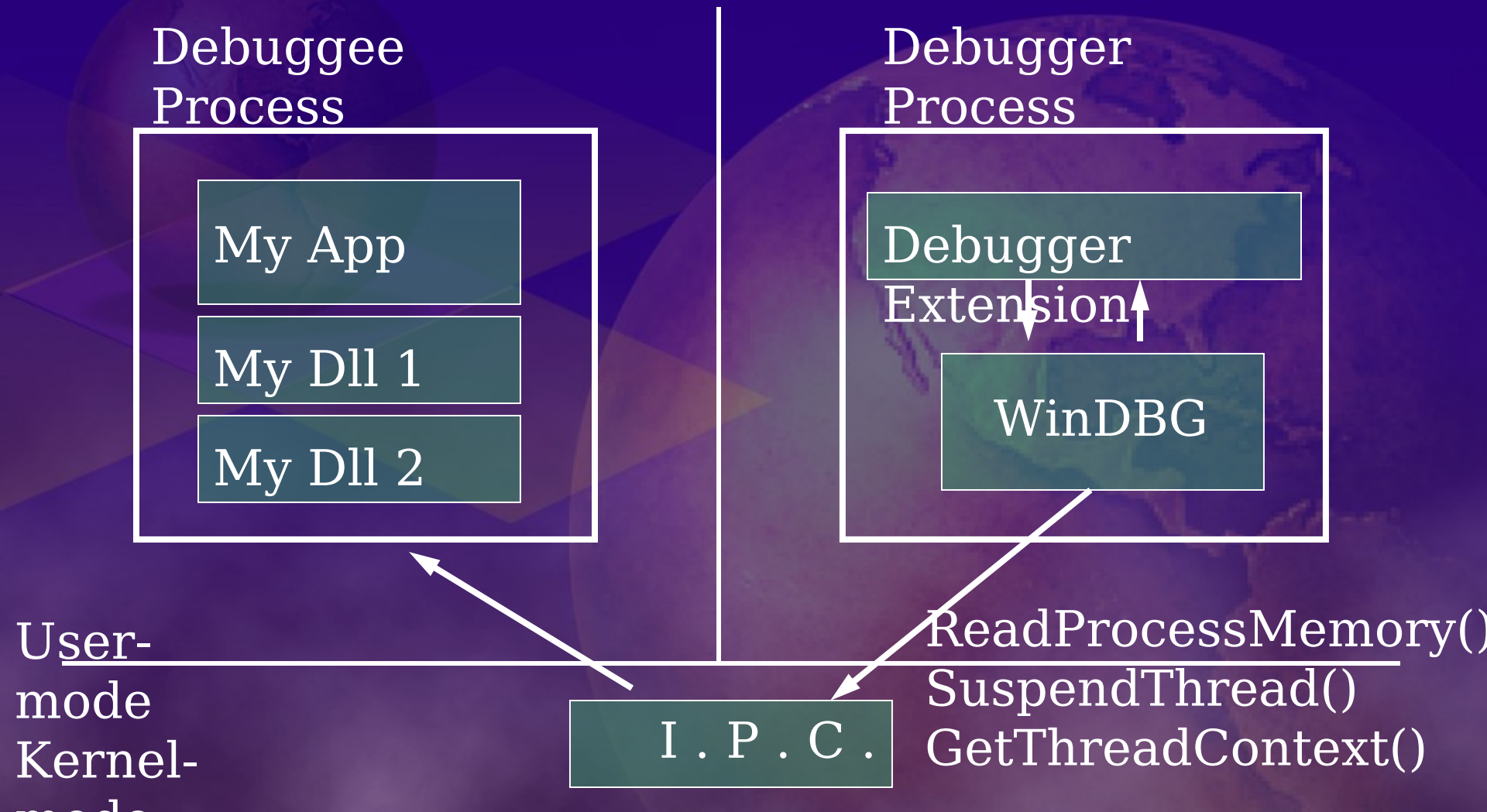
New Debugger Interfaces

- Initialization
- Read and write memory
- Process \ thread information
- Breakpoint
- Execution control
- Expression evaluator
- Full symbol and TYPE support
- Source code support

Debugger Extension Execution Model

- Debugger extensions are DLLs loaded by the debugger using LoadLibrary
  - Debugger extensions run in the context of the debugger process (WinDBG)
- Debugger can call :
  - Debugger interfaces (dbgeng.dll)
  - Any other Win32 API in the context of WinDBG

Debugger Extension Execution Model



Sample Debugger Extension

- Tool to remotely connect to a debug session and retrieve the stack trace

```
Main(pszConnectString) {
      DebugConnect(pszCS,IID_D, &gDbg);
      gDbg->QueryInterface(IID_C, &gControl);
      gDbg->SetOutputCallbacks(&gOutCallback);
      gDbg->ConnectSession(
            DEBUG_CONNECT_SESSION_DEFAULT,10);
      gControl->Execute(“k”); }

OutputCallback::Output(Mask, Text) {
      // handle output appropriately
}
```

Writing Debugger Extensions

- Package has an SDK component
  - have headers, libs and samples
- Write debugger extensions – they will make debugging simpler

Real Example Extensions

- AppCenter team runs into certain know 1st chance AVs during stress
- Solution:
  - Run the app under the debugger
  - Write a special extension to handle exceptions
  - For each AV, the debugger extension checks the address against a built-in list of known 1st change AVs
  - If it's a known AV, automatically continue the app

32 bit vs. 64 bit extensions

- All new interfaces use 64 bit addresses
- Debugger extensions should always use 64 bit addresses
- 32 bit addresses are SIGN-EXTENDED
  - Must handle this correctly in extension DLLs

64 bit Debugging

What's different with 64 bits

- Not that much …
  - Addresses, pointers are 64 bit. You must become familiar with the size of the various types
  - Alignment on many things change
- IA64 is one of the 64 bit architectures
  - 64 bit != IA64
  - IA64 is new, VERY complex instruction set

IA64 debugging

- Good docs in the debugger package
  - IA64 architecture description
  - IA64 annotated disassembly
- Recommend doing source level debugging whenever possible

Remote Debugging

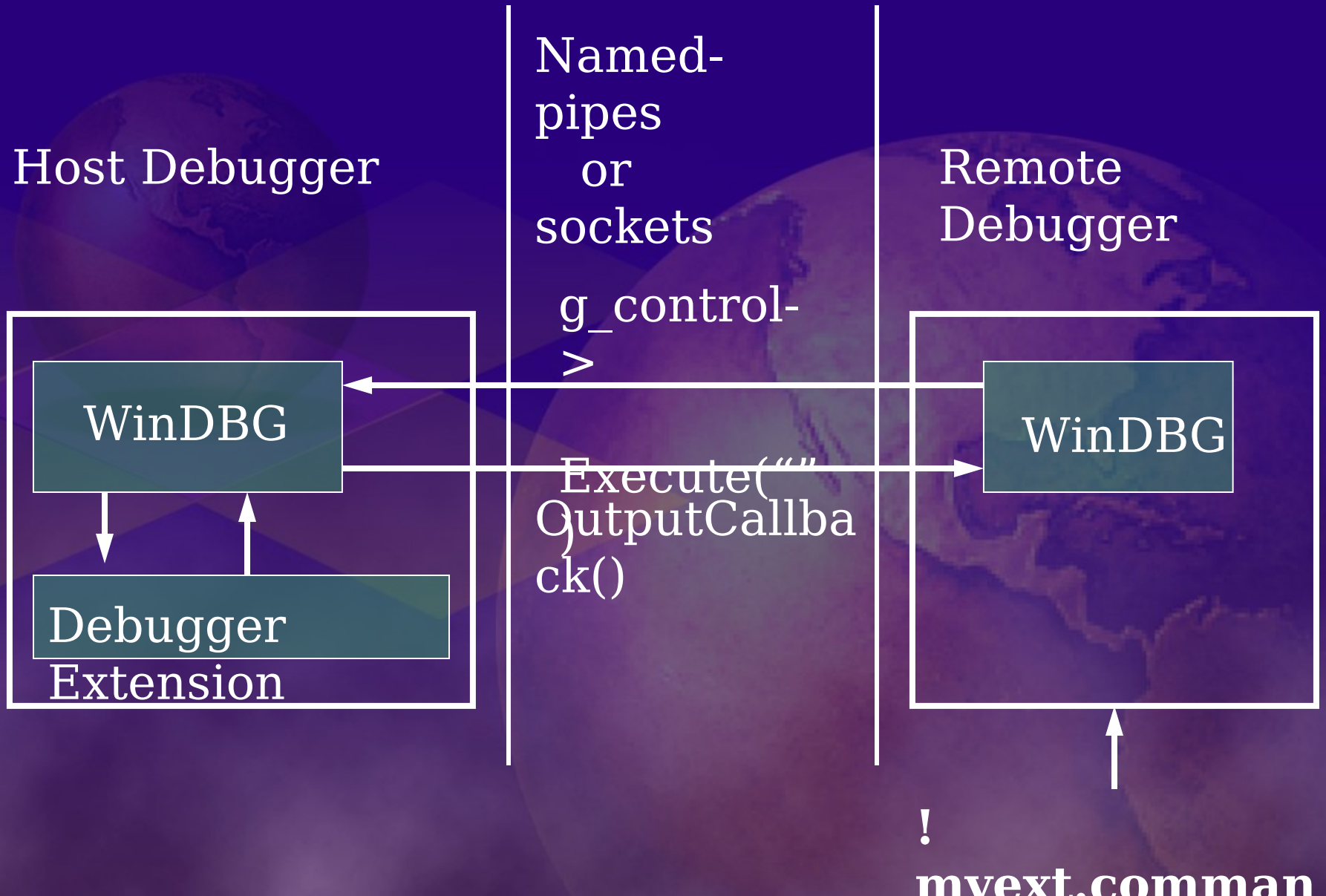
Remote Debugging

- Remote.exe
  - Console remoting program
  - Nothing to do with the debugger
    - You can remote a cmd.exe window
- New remote debugging support
  - Built-in, debugger specific protocol
  - Available in all the new debuggers
    - Windbg, kd, cdb, ntsd
- Built-in remoting and remote.exe do not talk to each

Remote Debugging

- Can connect remotely to an existing debug session
- Support various protocols
  - Named pipes, sockets (1394 coming)
- Multiple remote debuggers can connect to the host debugger
  - Must use the same transport
- Cannot remotely connect directly to a target

Remote Execution Model



Action Items

- Give WinDBG a try
- **Read the docs**
- Let us know what you need

Contact Info

- **Web site:**
  - http://www.microsoft.com/ddk/debugging
- **Problems, Questions, Feature Requests**
  - **Help file (Debugger.chm) is in the debugger package**
  - mailto:windbgfb@microsoft.com